# Network Court Protocol and Malicious Node Conviction

Na Li and David Lee
Department of Computer Science and Engineering, The Ohio State University
lina,lee@cse.ohio-state.edu

*Abstract* – A Network Court Protocol is designed for malicious node conviction based on the information from network node accusing and testifying operations, which are formally modeled by algebraic operators. It is shown that the malicious node unambiguous conviction is equivalent to the uniqueness of the solution of a system of Boolean equations and that is equivalent to the uniqueness of a corresponding satisfiability problem. A linear time algorithm is presented for the conviction process using a conviction graph search. The general protocol and algorithms are applied to P2P networks with promising experimental results.

Index Terms – Malicious node conviction, Court protocol, System of Boolean equations, Satisfiability, Conviction graph, P2P networks, Ring conviction protocol

## I. Introduction

Malicious nodes disrupt network operations and pose a serious security threat, particularly in distributed application environment, such as peer-to-peer (P2P) [L06, P02, T04] and overlay networks [Ra01, Ro01, S01, Z01]. For instance, a malicious node can forward a false lookup request or erroneous routing update, corrupt information stored in the system by repeatedly joining and leaving the network, launch an attack on a specific data item by ID mapping [S04], or exploit millions of concurrently interactive peers as an engine for DDoS attacks [N06].

A lot of efforts have been devoted to detecting malicious nodes by their abnormal behaviors, such as generating abnormal traffic. Packet payload can be checked for abnormality [M03]. Buffer overflow attack (a popular scheme for large scale attacks such as worms, zombies, or a large number of hackers running attack scripts) can be detected by identifying anomalous tokens in packet payloads, e.g., byte strings resembling injected jump addresses [L05]. In addition to the packet content, packet statistics [B01, W04, W05] and traffic flows [La04, Le04] can also reveal node abnormal behaviors. Port scans or DoS attacks can be detected by monitoring network parameters, such as the number of connections established and packet transmission rate [Kr02]. Analysis of multivariate time series of byte counts, packet counts, and IP-flow counts are used to illustrate and classify the anomalies, including DoS attacks, flash crowds, port scanning, downstream traffic engineering, high-rate flows, worm propagation, and network outage [La04].

We have briefly described a few network node abnormal behaviors with no intention to provide a complete survey; this paper is not on node abnormal behavior detection. Instead, our goal is to investigate the decision making based on the reports of network abnormal behaviors. Specifically, in a distributed environment, nodes may alert the presence of attacks - some are true and some are false – and we have to make a decision to identify the malicious nodes.

Formally, based on the observed behaviors or its own malicious intention, a node can make a statement on any other node as being malicious or good, or claim having no knowledge of it. Based on the statements of all the nodes, a protocol [G98] is needed to identify and eject the truly malicious nodes. This is similar to court activities. Defendants, prosecutors, and witnesses can accuse or testify for another person, or claim having no knowledge at all. Based on the statements, the jury or judge have to reach a verdict to convict a criminal, or to prove his innocence, or to quit the case for lacking of evidence.

The malicious node identification problem has been studied in the published literature. Douceur and Howell [D06] uses Byzantine distributed algorithm to isolate faulty nodes. Fireflies [J06] proposes a Byzantine tolerant solution to monitor stopping failure by multiple rings. Reputation [D04, G06, S05, S06] and Ranking systems [Y07] are used in P2P networks to leverage users' assessment of others for ranking a peer as trustworthy or for resource allocation. Morselli et al. explore the client verification [M04].

This work is to design a protocol for convicting malicious nodes in a distributed network environment based on the statements of the nodes on other nodes; a node can accuse another node as being a malicious one or testify for its integrity. Apparently, it is not an attack detection procedure. It is not network element authentication either where the goal is to verify the identity of the element being authenticated [Ka02]. It is not a verification procedure as [M04] or fault tolerant assessment [A98, E04]. It is different than Byzantine agreement where algorithm is used to permit a collection of processors to reach consensus [L86] in the presence of Byzantine failures of some processors [L96]. It is not a reputation system that collects, distributes, and aggregates feedbacks about participants' behaviors for determining their level of trust [R00]. We call it a *Network Court Protocol* for convicting malicious nodes.

Malicious node conviction is a challenge, particularly, in a distributed network environment. The complication is similar to that in a court: a criminal may falsely accuse an innocent person or testify for another criminal. Similarly, a malicious node can falsely accuse good nodes, or testify for other malicious nodes, or accuse a malicious node to make the conviction harder, or may even behave non-deterministically.

Formally, a node can make one of the following statements with regard to another node: (1) Accuse it as a malicious node; (2) Testify it is a good node; or (3) Claim having no knowledge of it. The node conviction process is to classify nodes as good or malicious based on the statements by all the nodes.

In general, this is an unsolvable problem. For instance, if there are only two nodes who accuse each other, there is no way one can tell whether both are malicious or one is good and the other is malicious. We make the following natural assumption:

*Assumption 1.*The statement a good node can make on another node:
(1) Testifying it is a good node; the testified node must also be good.
(2) Accusing it is a bad node; the accused node must be bad.
(3) Claiming no knowledge of it; it can be either good or bad. []

*Remark 1.* Assumption 1 is very general:
(1) A node can make a statement on any other nodes – they may or may not be its neighbors in a network.
(2) A good node never accuses a good node nor testifies for a bad node; a good node is reliable and trustworthy.
(3) There is no assumption on malicious node behaviors: it can testify for, accuse, or claim no knowledge of a good or bad node. []

In Section II, we formally model and formulate the problem. The Court Conviction Protocol is designed in Section III with a linear time algorithm for an unambiguous conviction. The general theory is applied to P2P networks with experiments in Section IV and V. Section VI contains remarks on variations and generalizations of the Conviction Problem.

## II. Mathematical Model

For simplicity, we call a malicious node a bad node, denoted by a Boolean value 0, and call a healthy node a good node, denoted by 1. As described before, each node can make a statement on another node: (1) Accusing it as a bad node; (2) Testifying for it as a good node; or (3) Claiming no knowledge of it.

We use an operator $\otimes$ to denote accusation; $x \otimes y$ for node $y$ accusing node $x$. By Assumption 1, a node must be bad if it is accused by a good node, and we have: $* \otimes 1=0$ where * stands for 0 or 1. On the other hand, a node does not change its nature if it is accused by a bad node, and we have: $* \otimes 0=*$. In summary, we have the following truth table:

| $x$ | $y$ | $x\otimes y$ |
|---|---|---|
| $0$ | $0$ | $0$ |
| $0$ | $1$ | $0$ |
| $1$ | $0$ | $1$ |
| $1$ | $1$ | $0$ |

It can be easily shown:

$$x \otimes y = x \,.\, y' \qquad (1)$$

where " . " is the Boolean "and" operator that is often omitted whenever there is no confusion, and " ' " represents the complement.

We use an operator $\oplus$ to denote testify-for; $x \oplus y$ for node $y$ testifying for node $x$. By Assumption 1, a node must be good if it is testified by a good node, and we have: $* \oplus 1=1$. On the other hand, a node does not change its nature if it is testified by a bad node, and we have: $* \oplus 0=*$. In summary, we have the following truth table:

| $x$ | $y$ | $x\oplus y$ |
|---|---|---|
| $0$ | $0$ | $0$ |
| $0$ | $1$ | $1$ |
| $1$ | $0$ | $1$ |
| $1$ | $1$ | $1$ |

It can be easily shown:

$$x \oplus y = x + y \qquad (2)$$

where "+" is the Boolean "or" operator.

A "no knowledge" statement does not have any impact on the target node and there is no need of any operator for it. Note that the accusation operator is not commutative but the testify-for operator is.

We now formally define the problem. We choose to present it as a general problem in distributed computing; its application to networking is a special case. Again Boolean value 0 and 1 represent bad and good element, respectively.

*Court Conviction Problem*

Given a set of $n$ elements of unknown Boolean values $x_i$, $i=1,...,n,$ a set of $u$ accusation statements: $x_{i_p} \otimes x_{j_p}$, $p=1,...,u,$ and a set of $v$ testify-for statements: $x_{i_q} \oplus x_{j_q}$, $q=1,...,v,$ where $\otimes$ and $\oplus$ are the accusation and testify-for operator, respectively, determine the value of each element.

The problem is *completely solvable* if the value assignment to the elements is unique. []

When the problem is completely solvable all the bad elements are uniquely identified and unambiguously convicted and all the good elements are also identified. Otherwise, the good and bad elements cannot be all identified unambiguously.

From Eq. (1) and (2), we can rewrite the statements in Court Conviction Problem as a system of $u+v$ equations of $n$ Boolean variables:

$$x_{i_p} = x_{i_p} x_{j_p}', \quad p=1,...,u \qquad (3)$$

$$x_{i_q} = x_{i_q} + x_{j_q}, \quad q=1,...,v \qquad (4)$$

This system has at least one solution that is the true value of the elements, that is, 0 for bad and 1 for good element. If the system has a unique solution, then the Conviction Problem is completely solvable: the solution is the same as the true values of all the elements.

We have formulated a very general bad element Conviction Problem. For our particular application of network malicious node conviction, an element is a network node, an accusation is from an attack alert, and a testify-for operation is to prove another node is good. Note again that an accusation or a testify-for operation by a node may not be on its immediate neighbors in the network.

We can use a *conviction graph* to represent all the accusation and testify-for operations. An $a$-edge $x \xrightarrow{a} y$ indicates that $x$

accuses $y$, and a $t$-edge $x \xrightarrow{t} y$ represents that $x$ testifies for $y$. No edge between nodes implies no knowledge.

*Example 1.* Figure 1 contains a conviction graph with node A=1, known to be good a priori. Node B is testified by A and hence is also good. By a same argument C is also good. D must be bad since otherwise C would be bad by the $a$-edge DC. Therefore, there is a unique solution: A=B=C=1 and D=0. If we remove the $a$-edge DC, D can be either good or bad – more than one solution.

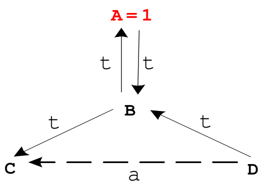


**Fig. 1** A Simple Conviction Graph

The Court Conviction Problem is now reduced to solving a system of Boolean equations. It has at least one solution: the true value of the nodes. The problem is *completely solvable*, that is, the good and bad nodes can be uniquely identified, if and only if it has a unique solution. There are a number of publications on the problem of uniqueness of solutions of systems of Boolean equations [P57, T66, W01] in which the necessary and sufficient conditions of uniqueness of solutions of general systems of Boolean equations are given yet there are no efficient algorithms provided. We explore the structure of the system of Boolean equations from the Court Conviction Problem and present linear time algorithms.

It is well known [M72] that two Boolean variables $x=y$ if and only if $xy + x'y' = 1$. For equation x=xy', we have $xxy' + x'(xy')' = 1$. A straightforward computation yields: $x' + y' = 1$. Similarly, for equation $x = x + y$, it is equivalent to: $x + y' = 1$. Therefore, we can rewrite Eq. (3) and (4) as:

$$x_{i_p}' + x_{j_p}' = 1, \quad p=1,...,u \tag{3'}$$

$$x_{i_q} + x_{j_q}' = 1, \quad q=1,.., v \tag{4'}$$

Therefore, the system of Boolean equations (3) and (4) is equivalent to the following:

$$\prod_{p=1,q=1}^{p=u,q=v} (x_{i_p}' + x_{j_p}')(x_{i_q} + x_{j_q}') = 1 \tag{5}$$

Obviously, it is a satisfiability problem (SAT) [H79] of the Boolean expression on the left side. We conclude:

*Proposition 1.* The system of Boolean equations (3) and (4) has a solution if and only if the Boolean expression on the left side of (5) is satisfiable, and the system in (3) and (4) has a unique solution if and only the Boolean expression in (5) is uniquely satisfiable. []

*Corollary 1.* The Court Conviction Problem is completely solvable if and only if the system of Boolean equations (3) and (4) has a unique solution, and this is the case if and only if the Boolean expression in (5) is uniquely satisfiable. []

## III. NETWORK COURT CONVICTION PROTOCOL DESIGN

The Court Conviction Problem is now reduced to the satisfiability problem and its unique solution. It is well known that the general satisfiability problem is NP hard and so is its uniqueness [P94]. However, each clause in (5) only contains two Boolean variables and it is a 2-SAT problem and there are polynomial time solutions [P94, V03]. We now present a linear time algorithm using the conviction graph.

From a Conviction Problem defined in (3') and (4'), construct a conviction graph as in Fig. 1. Each $a$-edge from node $y$ to $x$, $y \xrightarrow{a} x$, is associated with a clause of the Boolean expression in (5): $x' + y'$, which must have value 1in the conjunctive normal form of (5). Similarly, each $t$-edge $y \xrightarrow{t} x$ corresponds to a clause $x + y'$ in (5) with value 1.

It can be easily checked that $x + y' = 1$ and $x' + y = 1$ leads to $x = y$. Therefore, if a pair of nodes testifies for each other, then they must be of the same kind: both are bad or good. This introduces an equivalence relation on the nodes and we can first merge the equivalence classes of nodes in the graph.

Obviously, the Conviction Problem has a trivial solution: all the nodes are bad (all variables have value 0). We rule out this uninteresting case and assume that there is at least one good node (if all the nodes are compromised, the network is hopeless). We search the conviction graph in two phases for solving the Conviction Problem as follows.

*Search 1.*

Starting from each of the good nodes, conduct a search (depth-first or breadth-first) of the conviction graph along $a$-edges and $t$-edges as follows. Suppose that we are searching an edge from node $y$ to $x$:

(1) $y=1$ and $y \xrightarrow{a} x$. Since $x' + y' = 1$, $x=0$. Node $x$ is accused by a good node $y$. If x has not been assigned a value yet, assign it 0 - a bad node. Otherwise, $x$ has been assigned a value, and there are two cases, if $x$ has a value 0 – consistent, back track from the edge. Otherwise, a contradiction – abort the process.

(2) $y=1$ and $y \xrightarrow{t} x$. Since $x + y' = 1$, $x=1$. Node $x$ is testified by a good node $y$. If $x$ has not been assigned a value yet, assign it 1 - a good node. Otherwise, $x$ has been assigned a value, and there are two cases. If $x$ has a value 1 – consistent, backtrack from the edge. Otherwise, a contradiction – abort the process.

(3) $y=0$ and $y \xrightarrow{a,t} x$. Since $x' + y' = 1$ ($a$-edge) or $x + y' = 1$ ($t$-edge), $x$ can have either values 0 or 1 in both cases; the value of $x$ cannot be decided and the search from it cannot continue; back track from the edge. Consequently, the search always backtracks from a node with value 0 – a 0-node.

When the search is completed if all the nodes are assigned a value then we have a unique solution. However, if there are nodes with undecided values (not searched), we cannot claim the uniqueness of the solution, and we have to further process these nodes in Search 2 next.

*Search 2.*

We denote the nodes with undecided values as β-nodes and call the subgraph of β-nodes a *residual graph*. From Case (3) of Search 1, an *a*- or *t*-edge from a 0-node has no impact on the end node, and we can assign 0 to all β-nodes and obtain a solution for all the β-nodes – a trivial solution for the residual graph with all 0-nodes. However, this may not be the only solution.

The conviction graph in Fig. 2 has a predetermined good node A=1. Following Search 1, we have B=1 and C=0. The remaining nodes cannot be determined with D=E=F=G=H=I=J= β, forming a residual graph.

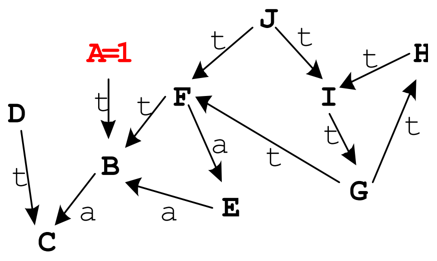


**Fig. 2** A Conviction Graph with β-nodes

One can assign 0 to all the β-nodes in the residual graph – a trivial solution for it - and obtain a *non-trivial solution* of the original Conviction Problem. Therefore, the original problem has a unique solution if and only if the residual graph only has a trivial solution, that is, none of the β-node can be assigned value 1 without introducing any contradictions.

Note that some β-nodes of the residual graph may be adjacent to 0- or 1-nodes via *t*- or *a*-edges, such as D, E and F nodes. We can repeat the following process on each β-node. Assign value 1 to it and perform the search on the residual graph using Search 1. If the search is completed without any contradiction, then we have a non-trivial solution on the residual graph by assigning 0 to all the remaining β-nodes; we have obtained a non-trivial solution for the residual graph and hence the original conviction problem has more than one solution – not completely solvable. However, as in Search 1, a search to a 0- or 1-node may introduce a contradiction. In this case, we reverse the value of that start node to 0 (it cannot have value 1), reverse other assigned values back to β, select another β-node, assign value 1, and repeat the same process. We continue until: (1) Obtain a non-trivial solution on the residual graph – the original Conviction Problem does not have a unique solution; or (2) All the β-nodes are assigned value 0, a trivial solution, and hence the original problem has a unique solution. However, in the worst case, each trial of a start node takes time proportional to the number of edges and there are *n* trials; the total cost is quadratic. We now present a linear time algorithm for processing the residual graph.

Note that the search backtracks when encounters a 0- or 1-node; when a node is assigned value 0 by the search, we backtrack from it as well, as illustrated in (3) of Search 1. Therefore, only searching from a 1-node is consequential.

Starting from an arbitrary β-node, conduct a depth-first-search of the residual graph of β-nodes along t-edges, and shrink each SCC (Strongly Connected Component) into a β-node, obtaining a DAG (Directed Acyclic Graph). Obviously, all the β-nodes in an SCC must have a same value 0 or 1, since they testify for each other. Topologically sort the DAG and each node is assigned an integer from sorting [C01].

We now examine the residual graph with both *t*- and *a*-edges (A shrunk node from an SCC inherits all the incoming and outgoing edges.) Take the β-node with the smallest sorting number; it has incoming edges from other β–nodes but does not have any outgoing edges to any other β-nodes. On the other hand, it may have *t*- and *a*-edges to or from nodes with assigned value 0 or 1, which are "outside" the residual graph. We now attempt to assign value 1 to the selected β-node. If it causes a contradiction; for instance, it has an *a*-edge going to a node with value 1 (accuse another good node), then we can only assign value 0 to the selected β-node, and we do assign 0 to it. We then repeat the process on the next β-node with the smallest sorting number. However, if there is no contradiction from assigning 1 to the selected β-node, we assign 1 to it and 0 to all the remaining β-nodes in the residual graph. We claim that there is no contradiction and we obtain a non-trivial residual graph since the selected β-node is assigned value 1. Indeed, since all the other nodes in the residual graph have value 0, they will not cause any contradiction. The only possible contradiction is: there is a *t*-edge (*a*-edge) from a 1-node outside the residual graph to a node in the residual graph that has value 0 (1 – the selected β-node). This is impossible; otherwise, this edge would have been searched during Search 1, and the end node under consideration would not be in the residual graph. Therefore, the original problem does not have a unique solution and we terminate the process.

Therefore, we either terminate the assignment process upon obtaining a non-trivial solution, or we assign 0 to all the β-nodes in the residual graph. In the former case, the original problem does not have a unique solution, and in the latter case, the residual graph has a trivial solution and the original problem has a unique solution.

Obviously, each edge in the residual graph, including the ones coming in and going out of the graph, is examined a constant number of times: depth-first-search for obtaining a DAG, topological sort, and checking each node bottom-up, and the total cost is proportional to the number of edges.

In summary,

*Theorem 1.* The Court Conviction Problem is completely solvable if and only if the corresponding system of Boolean equations (3) and (4) has a unique solution if and only if the corresponding Boolean expression in (5) is uniquely satisfied, and this is the case if and only if the search of the Conviction Graph assigns each node a unique Boolean value. The total cost of the conviction graph search is O($m$) where $m$ is the number of edges, and hence the Court Conviction Problem can be solved in time proportional to the total number of accusations and testifies. []

*Algorithm 1* (Conviction Graph Search)
Input: Conviction Graph G= (V, E)
Output: Conviction Problem has a unique solution or not with array A[$u$], $u$=1, 2…, $n$=|V| to indicate node value assignments: 1, 0, or β for good, bad and undetermined node, respectively.

```
Queue $Q \leftarrow \varnothing$ ; /* nodes with determined value 1 */
List $L \leftarrow \varnothing$ ; /* nodes of unknown value */
for $u$=1 to $n$
   if node $u$ is known to be good a priori
      A[$u$] = 1, ENQUEUE ($u$, $Q$);
   else /*  node $u$ value is unknown */
      A[$u$] = β, insert $u$ to $L$;
Search 1:
   while $Q \neq \varnothing$
      $u$=DEQUEUE($Q$);
      for each $v \in Adj(u)$ /* end node of edge from $u$ */
         if A[$v$] = β
            if E($u$, $v$)=‘$a$’ /* $a$-type edge */
               A[$v$] = 0;
            else /* $t$-type edge E($u$, $v$)=‘$t$’ */
            A[$v$] =1; ENQUEUE($v$, $Q$);
            delete $v$ from $L$;
   if $L = \phi$ /* residual graph empty; search completed */
      return “Conviction Problem has a unique solution.”
   else /* search residual graph of β-nodes in $L$ */
Search 2:
      Examine $t$-edges only in residual graph, merge SCCs,
      conduct a topological sort, obtain a DAG with β-nodes
      listed in $L'$ in sorted order;
      While $L' \neq \varnothing$
         $u$=DELIST ($L'$) /* examine β-nodes bottom up
                                    in topological order */
         for each $v \in Adj(u)$ /* end node of edge from $u$ */
            if (E($u$, $v$) = ‘$a$’ and A[$v$]=1) or
               (E($u$, $v$) = ‘$t$’ and A[$v$]=0)
                $u$=0, exit; /* cannot assign value 1 to $u$ */
         $u$=1; /* $u$ can be assigned value 1 */
         return “Conviction Problem has no unique solution.”
      return “Conviction Problem has a unique solution.”
[]
```

*Remark 2.*

1. The Conviction Problem is completely solvable if and only if Algorithm 1 returns “Conviction Problem has a unique solution.” In this case, all the good and bad nodes are uniquely identified: A[$v$]=1: $v$ is a good node; A[$v$]=0: $v$ is a bad node, $v$=1,2,…,$n$.
2. For completeness, we remark on the general case that there is no a priori information of any nodes – all the nodes are of β-type. The trivial solution is to assign value 0 to all of them without introducing any contradiction. The question is: are there any non-trivial solutions? If the answer is no, there is a unique solution – all the nodes are compromised, an uninteresting case.

This is exactly the same problem as Search 2 of the residual graph except that no β-node has an outgoing or incoming edge from a node with an assigned value. We can examine $t$-edges, shrink SCCs, and conduct a topological sort of the resulting DAG. Examine the bottom node – an SCC. If there is any $a$-edge among any two nodes in the SCC, we can only assign 0 to all the nodes in the SCC. Otherwise, we can assign 1 to all the nodes without introducing any contradiction. In the first case, we examine the next node (SCC) above and repeat the same process. In the second case, we have found a non-trivial solution and the Conviction Problem does not have a unique solution – not completely solvable. []

## IV. APPLICATION TO P2P NETWORKS

We apply the general malicious node Conviction Protocol to Peer-to-Peer (P2P) networks. P2P network routing follows a virtual ring that is formed dynamically from Distributed Hash Table (DHT) [A06] and that has also been proposed for general routing [C06].

### *A. P2P Network and Virtual Ring*

For P2P applications in a distributed environment, it is more difficult to identify malicious peers than in a network with a centralized control and management. It is often hard to trace back to the original source of malicious behaviors; healthy nodes may act involuntarily as accessories of malicious behaviors. For instance, for the resource lookup service in a P2P network [S01], when a malicious node launches a DDoS attack, it injects false information for attracting traffic to a victim node by informing the whole network that the victim has the needed resources. This false information may have been propagated to a large domain when the attack is detected and a number of healthy nodes have participated in propagating the false information [W02].

We now apply the Conviction Protocol Algorithm 1 to virtual rings of P2P networks in a distributed way, and present a Virtual Ring Conviction Protocol.

Since P2P network traffic flows along a virtual ring, each node can monitor the behaviors of both its neighbors on the ring and detect anomalies. For instance, for the overlay lookup service [S02], when a node receives a request for resource it transmits the request to a neighboring node whose logical identifier is closest to the logical identifier of the resource. A malicious node may forward the request to an incorrect (not the closest) or non-existing node. This malicious behavior can easily be detected by its next hop logical neighbor on the virtual ring by checking whether the request is getting “closer” to the resource identifier.

### *B. Malicious Node Conviction*

We now study malicious node Conviction Problem on this virtual ring network, using the general Algorithm 1 in Section III. Every node in the ring has two neighbors, a clockwise one and an anticlockwise one. As a case study, we assume that a node can only testify for or accuse its neighboring node on the ring. Apparently, the conviction graph is the ring itself with $t$- and $a$-edges between immediate neighboring nodes. This special case has been studied in [L07] informally and with strong assumptions. We also assume that a node either accuses or testifies for a neighboring node, that is, it cannot claim no knowledge of it, after transmitting for and monitoring its behaviors. Needless to say, we can model or make assumptions differently for different applications and implementation environments and the general theory still applies. We now proceed with the P2P ring network case study.

As explained, if two neighboring nodes testify for each other, they must be of the same type –both are good or bad, and we can merge them into one node for the conviction process. Consequently, for two neighboring nodes A and B on the ring, there are two cases: (1) Both accuse each other: at lease one is bad; or (2) Node A accuses B but B testifies for A in return: B must be bad but A can be either good or bad. (If B were good, A must be good since B testifies for it, but A cannot accuse B – a contradiction). Based on the neighboring nodes classification, we consider the following three types of rings. We state the results, which were obtained in [L07] by lengthy proofs but still incomplete. With Algorithm 1 we can easily conclude:
Case 1. All the edges are of Type (1). It has multiple solutions: the trivial solution with all the nodes bad; and any assignment with good nodes isolated by bad ones.
Case 2. All the edges are of Type (2) but may in different directions. Consider *a*-edges only on the ring. All the source nodes can have value either 0 (bad) or 1 (good), and all the remaining node must have value 0 (bad).
Case 3. The ring contains both types of edges. We can segment the ring to parts with Type (1) or (2) edges and determine accordingly.

Obviously, the Conviction Problem does not have a unique solution in general. One would explore practical constraints for convicting bad nodes. Various specific assumptions were made in [L07] for obtaining convictions with detailed proofs. The interested readers are referred to [L07] for details.

As a case study, we consider the situation that bad nodes are isolated, that is, there are no two neighboring bad nodes. This is a typical scenario before bad nodes take over the network; they are still in isolation in the ring. If we can run Conviction Protocol frequently and fast enough, we can eject the malicious nodes in a timely manner and maintain a healthy ring. As reported in Section V, this is often the case in practice.

If there are only *t*-edges, then all the nodes are good. If there is at least one *a*-edge, we examine the edges on the ring of Type (1) and (2). If there are only Type (1) edges, due to the bad node isolation condition, the only possible cases are: there is an even number of nodes and good/bad nodes are interleaved, and in this case, there are only two solutions. Note that if there is an odd number of nodes on the ring, this is impossible. We now consider the last interesting case: there are mixed Type (1) and (2) edges.

For an *a*-edge in a Type (2) edge, the end node must be bad, and by the bad node isolation condition, its two neighbors must be good. Starting from a good node, its unexplored neighbor is good or bad, depending on it is testified for or accused by the good node. If the newly determined neighbor is good, we can repeat the process. If it is bad, then the next unexplored neighbor must be good by the bad node isolation condition. We can repeat the process until all the nodes are uniquely identified: convicted or cleared unambiguously.

As illustrated in [L07], without the bad node isolation condition often the Conviction Problem can still have a unique solution with other constraints. We will not attempt to exhaust all possible cases here; the interested reader may apply the general Algorithm 1 and use the similar approaches to investigate their application problems with particular assumptions and constraints.

## *C. Virtual Ring Conviction Protocol*

Using the Conviction Algorithm 1, we now describe a malicious node conviction protocol on a virtual ring from P2P networks.

The protocol utilizes a *token* for collecting and distributing information – accusation or testifying for – for making a conviction decision in a distributed way.

The token contains two fields (bits) for each node on the ring to evaluate its two neighbors; it can testify for (assert TURE) or accuse (assert FALSE) its clockwise neighbor and anticlockwise neighbor according to its diagnosis of the two neighbors from monitoring. The token is passed around in one direction - clock-wise only (for clarity). The token is passed around the ring twice before the conviction process starts: first time for collecting information and the second time for distributing the collected information.
*Phase 1 (information collection):* The token is passed along the ring clockwise to collect the accusation/testify-for information by all the nodes. A node can only access to its own evaluation fields (2 bits in total) during this phase.
*Phase 2 (information distribution and Conviction):* In this phase, all the fields in the token are read-only to all the nodes: visible but unchangeable. The token is passed along the ring for a second round to distribute the information. Upon examining information in the token, each node runs Conviction Algorithm; it either uniquely identifies all the good and bad nodes or claims the Conviction Problem is not completely solvable.

## *D. Token Security Management*

In the above two phases, cryptographic techniques are needed to guarantee that the token is securely passed along the ring without being viewed by an unauthorized node nor altered. We can use public key scheme or hash chain for authentication as done in secure routing protocols [H03].

However, if one or more malicious nodes persistently scramble the whole token content or – even worse – drop the token, then there is not much one can do in a same layer. This problem is different than that in the usual token ring protocol where one has to deal with the accidental loss or scrambling of tokens. One might consider transmitting token in a different layer in a secure way with a new protocol or with a cross-layer approach. It is an interesting problem yet is not a topic of this work. Without further digressing we assume that tokens can be passed along the ring without being scrambled or dropped.

Assume that there are *n* nodes on the ring (in the network). We pass the token all over the ring twice for collecting and distributing the information with a cost O(2*n*). Upon receiving the token the second time, each node – good node rather - detects bad peers. Since the Conviction Algorithm runs in linear time, we have:

*Proposition 2.* It takes time proportional to the number of nodes in a ring network to run the Conviction Protocol to either uniquely convict the bad nodes or to conclude the problem is not completely solvable. []

## V. Experiments

We implement the Conviction Protocol Algorithm 1 and apply to a ring of a P2P network that is constructed on PlanetLab testbed [PL02]. It is a global research network consisting of nodes deployed all over the world. We select 50 nodes located in a wide geographical region, including North American, Asia and Europe. These nodes are organized into a virtual ring according to DHT and every node can accuses or testifies for its two immediate neighbors. For the experiments, every node makes a random decision on its neighbors.

### A. *Performance*

Every five minutes a new token is generated by a random node in the ring and then going through the two Phases as in Section IV.C. The whole process is repeated for 24 hours and we record the time spent for each token to go through the two phases – called a round trip for convenience:

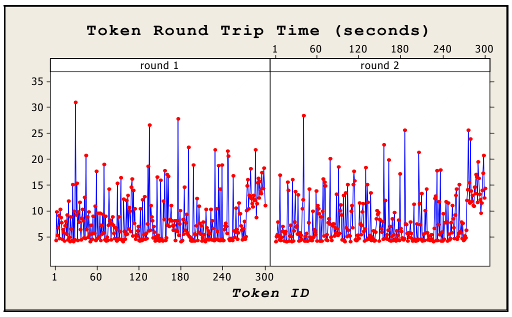


**Fig. 3** Token Round Trip Time

Figure 3 shows how long it takes for a token to be passed around the self-organized ring with 50 PlanetLab nodes; the time for a round trip - the two phases - is in a range of 4 to 32 seconds, and on the average it takes around 8 seconds. The token monitoring period is short enough that one can make a conviction based on updated information.

### B. *Impact of the Number of Nodes*

For scalability, it is important for the token round trip time to have a slow increase with the number of nodes in the ring. In figure 4, we observed an approximate linear increase for the average token round trip time versus the total number of nodes.

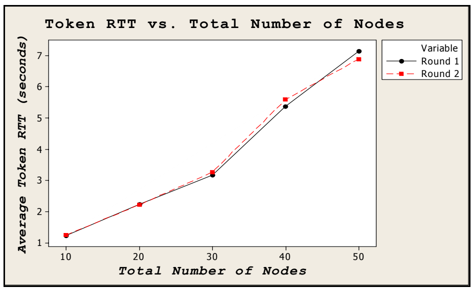


**Fig. 4** Linear Increase of Token Round Trip Time

### C. *Time for Token Security Management*

To guarantee that a token is securely passed along the ring without being viewed by an unauthorized node nor altered encryption/decryption or hashing can be used. Specifically, when a token is received, a node first decrypts it, obtains information in the token, makes its own judgment, inserts its statement, and encrypts the whole token before sending it back to the ring. In Section V.B we only measure ring passing time without including the token management time, since it depends on the node processing power and the cryptographic techniques used. In order to get a rough idea, we collect data of a typical setting of machines and crypto-techniques:

| | |
|---|---|
| **Encryption/Decryption** | AES |
| **Digital Signature/Verification** | RSA |
| **Key Length** | 128 bit |
| **Implementation Language** | C |
| **CPU** | Intel(R) Xeon(TM) CPU 2.66GHz |
| **Memory Size** | 2074924 k |

From Fig. 5, there is a linear increase of time from the total number of nodes in the ring with the token security management.

Token length may make a difference in time for the security management due to the encryption/decryption time. Figure 6 shows that token entry length has little impact on total time for token security management so long as each entry for a node has no more than 16 bytes (theoretically we only need two bits).

We provide a combined view of time for security management in Fig. 7 with 100 nodes in the ring. It shows that, compared with token round trip time, the time for token security management has more impact on overall performance of distributed token ring algorithm but still does not cause significant delay in processing.

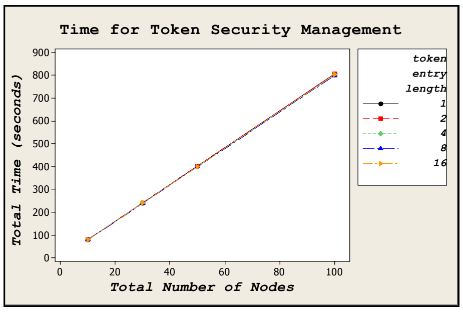


**Fig. 5** Time for Security Management vs. Number of Nodes

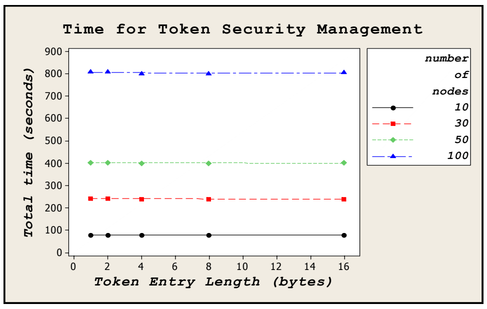


**Fig. 6** Time for Security Management vs. Token Entry Length

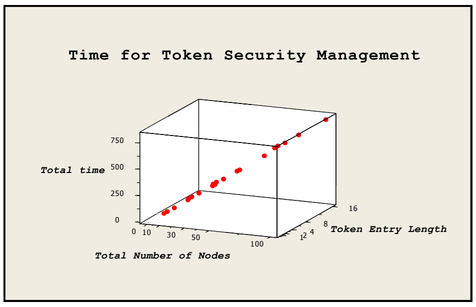


**Fig. 7** A combined view of Time for Security Management

### *D. Malicious Node Isolation Condition – an Practical View*

In Section IV we have shown that the Conviction Protocol has a unique solution on the ring with an assumption that malicious nodes are isolated, that is, we can convict and eject bad nodes in a timely manner when they are still in isolation. Is this a reasonable assumption in practice?

It depends on the rate/probability a nodes is being compromised. There is a variety of ways a node can be compromised, such as by attacks on an insecure service running on a host. These attacks are often launched by worms and can spread quickly [Wa02]. For example, in July 2001, Code Red infects 359,000 computers in less than 24 hours [CA01]; and in August 2003, MSBlaster infects 120,000 computers in 24 hours [B03]. However, as indicated in Fig. 8, the rate of compromise is very slow in the initial phase (less than 0.05%) for the first few hours.

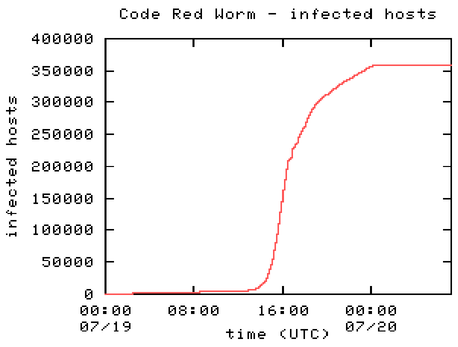
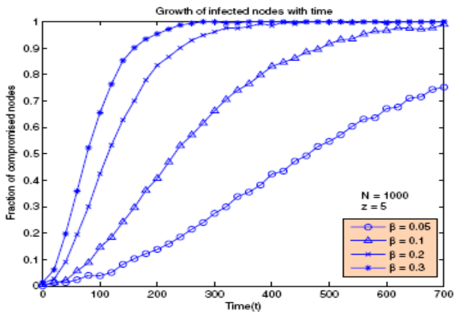


**Fig. 8** Observed Code Red propagation — number of infected hosts (from Caida.org)

Pradip De studied node comprise in wireless sensor networks using epidemic theory [De06]. As shown in Fig. 9, for a total of 1000 nodes with average node degree of 5, the fraction of compromised nodes are smaller than 0.05 within 20 seconds under different infection probability.



(a) Average node degree = 5

**Fig. 9** Node Compromise Modeling

Suppose that there are $n$ nodes in the ring and that it takes time T to execute the Conviction Protocol (passing the token twice along the ring and execute the protocol to reach a conviction). Suppose that the probability a node is compromised within time T is $p$. Then the probability that there is at least a pair of adjacent bad modes is no more than $np^2$ . From our experiments, $n=50$, $T=8sec$. For the case of Code Red infection, suppose that the compromise rate is *0.05%* in the first hour, we have $p = 0.05\% \times 8/3600 \approx 1.1\times 10^{-6}$, and we have $np^2 \approx 10^{-10}$. Therefore, the probability that the bad node isolation condition is violated is negligible. That is, if we constantly monitor the ring network and execute the Conviction Protocol, we can identify and eject the malicious nodes efficiently and in a timely manner for maintaining a healthy network.

## VI. CONCLUSION

We have presented a Network Court Protocol for malicious node conviction and investigated the necessary and sufficient conditions for the problem to be completely solvable – an unambiguous conviction. As a case study, we applied the general theory to P2P networks. There are a lot of issues and topics to be further explored and we briefly mention a few here.

(1) The Conviction Protocol can be either centralized or distributed. For instance, in a P2Pvirtual ring, each node can run the protocol and make a decision – a completely distributed approach. On the other hand, on an arbitrary network, a security administrator can collect the accusation and testify-for information from the network nodes and run the Court Protocol for convicting and ejecting malicious nodes – a centralized approach.

(2) For clarify, we present the protocol and algorithm based on information from all the nodes of a whole network. It is not necessary. One can run the Court Protocol based on the information available; there is no need to obtain information of the whole network; one can easily check that the modeling and algorithm still apply.

(3) We have been focused on whether a Conviction Problem is completely solvable, that is, whether each involved node can be uniquely identified as a good or bad one. So long as there is a node that cannot be uniquely determined, we claim that the problem is not completely solvable. As one can clearly see in

Algorithm 1 that after Search 1, a set of nodes are uniquely identified and that only the nodes in the residual graph may not be determined. The result of having uniquely identified a set of nodes provides a partial solution of the Conviction Problem – they are identified unambiguously, and we can eject the bad nodes from this process.

(4) The Court Protocol and Conviction Algorithm 1 are network topology independent; the accusation and testify-for information may or may not be on the neighbors of the acting nodes.

(5) The original Court Conviction Problem is formulated as a general problem in distributed computing and the network malicious node detection and P2P ring protocol are special applications. It may have applications in other areas.

(6) There is a variety of variations and generalizations of the Conviction Problem. For instance, the Court Conviction Protocol is based on Assumption 1, that is, good nodes are trustworthy and bad nodes can be as malicious as possible. In different application environments, the bad nodes can be more restrictive in their malicious behaviors or the good nodes might not be absolutely trustworthy – both assumptions can be relaxed. This will lead to a host of new problems for further investigation.

## Acknowledgment

This work has been supported in part by NSF awards CNS-0403342 and CNS-0548403 and by DoD award N41756-06-C-5541.

## References

[A06]B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. In Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, San Diego, CA, USA 2006.

[A98]A. Arora and S. S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In Proceedings of International Conference on Distributed Computing Systems, Amsterdam, The Netherlands 1998.

[B01]M. Bykova et al. Detecting network intrusions via a statistical analysis of network packet characteristics. In Proceedings of the 33rd Southeastern Symposium on System Theory, Athens, Ohio, 2001.

[B03]D. Becker and M. Hines. "FBI arrests MSBlast worm suspect." http://news.com.com/2100-1009-5070000.html.

[C01]T. H. Cormen et al. Introduction to Algorithms. 2001. ISBN: 0-262-03293-3

[CA01]CAIDA: http://www.caida.org/analysis/security.

[C06]M. Caesar et al. Virtual ring routing: network routing inspired by DHTs. In Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, Pisa, Italy, 2006.

[D04]P. Dewan and P. Dasgupta. Pride: peer-to-peer reputation infrastructure for decentralized environments. In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, 2004.

[D06]J. R. Douceur and J. Howell. Byzantine Fault Isolation in the Farsite Distributed File System. In Proceedings of the 5th International Workshop on Peer-to-Peer Systems, Santa Barbara, CA, USA, 2006.

[De06] P. De et al. Modeling Node Compromise Spread in Wireless Sensor Networks Using Epidemic Theory. In Proceedings of the 2006 international Symposium on World of Wireless, Mobile and Multimedia Networks, 2006.

[E04]A. Ebnenasir and S. S. Kulkarni. SAT-Based Synthesis of Fault-Tolerance. Fast Abstracts of the International Conference on Dependable Systems and Networks, 2004

[G01]M. G. Gouda. Elements of Security: Closure, Convergence, and Protection. Information Processing Letters, Vol. 77, pp. 109-114, 2001.

[G06]M. Gupta et al. Trade-offs between reliability and overhead in peer-to-peer reputation tracking. Computer Networks: The International Journal of Computer and Telecommunications Networking archive Vol.50, No. 4. 2006.

[G98]M. G. Gouda. Elements of Network Protocol Design. 1998. ISBN: 0-471-19744-0.

[H03]Y. Hu et al. Efficient Security Mechanisms for Routing Protocols. In Proceedings of the Tenth Annual Network and Distributed System Security Symposium, San Diego, CA, 2003.

[H79]Hopcroft and Ullman. Introduction to Automata Theory, Languages, and Computation. 1979. ISBN: 0-201-02988-X.

[J06]H. Johansen et al. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In Proceedings of Eurosys 2006, Leuven, Belgium, 2006.

[Ka02]C. Kaufman et al. Network Security: Private Communication in a Public World. 2002. ISBN:0-13-046019-2

[Kr02] C. Krügel et al. Service specific anomaly detection for network intrusion detection. In Proceedings of the 2002 ACM Symposium on Applied Computing, Madrid, Spain, 2002.

[La04]A. Lakhina et al. Characterization of network-wide anomalies in traffic flows. In Proceedings of the 4th ACM SIGCOMM Conference on internet Measurement, Taormina, Sicily, Italy, 2004.

[Le04]K. Levchenko et al. On the difficulty of scalably detecting network attacks. In Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington DC, USA, 2004.

[L05]Z. Liang and R. Sekar. 2005. Fast and automated generation of attack signatures: a basis for building self-protecting servers. In Proceedings of the 12th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 2005.

[L06]M. Li et al. DPTree: A Balanced Tree Based Indexing Framework For Peer-To-Peer Systems. In Proceedings of IEEE International Conference on Network Protocols, Santa Barbara, CA, 2006.

[L07]N. Li and D. Lee. Virtual Authentication Ring for Securing Network Operations. In Proceedings of First International Conference on New Technologies, Mobility and Security, Paris, France, 2007.

[L86]T. V. Lakshman and A. K. Agrawala. Efficient Decentralized Consensus Protocols. IEEE Trans. Software Eng. 12(5), pp.600-607, 1986.

[L96]N. A. Lynch. Distributed Algorithms. 1996. ISBN: 1-55860-348-4

[M03]M. Mahoney. Network traffic anomaly detection based on packet bytes. In Proceedings of the ACM SIGSAC, 2003.
[M04]R. Morselli et al. Trust-Preserving Set Operations. In Proceedings of IEEE Infocom'04, Hong Kong, 2004.
[M72]M. M. Mano. Computer Logic Design. 1972. ISBN 0-13-165472-1.
[N06]N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In Proceedings of the 1st international conference on Scalable information systems, Hong Kong, 2006.
[P02]S. W. K. Poon and J. Cao. Rheeve: A Plug-n-Play Peer-to-Peer Computing Platform. In Proceedings of IEEE Workshop on Resource Sharing in Massively Distributed Systems, Vienna, Austria, 2002.
[PL02]Planet Lab: http://www.planet-lab.org/.
[P57]W. L. Parker et al. On Uniquely Solvable Boolean Equations. The Journal of Symbolic Logic, Vol. 22, No. 1. pp. 96-97, 1957.
[P94]C. H. Papadimitriou. Computational Complexity. 1994. ISBN: 0-201-53082-1
[R00]P. Resnick et al. Reputation Systems. Communications of the ACM, 43(12), pp. 45-48, 2000.
[Ra01]S. Ratnasamy et al. A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, CA, USA, 2001.
[Ro01]A. Rowstron et al. Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. In Proceedings of IFIP/ACM Middleware 2001, Heidelberg, Germany, 2001.
[S01]I. Stoica et al . Chord: A Scalable Peer to peer Lookup Service for Internet Applications. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, CA, USA, 2001.
[S02]E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002.
[S04]M. Srivatsa and L. Liu. Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. In Proceedings of the 20th Annual Computer Security Applications Conference, Washington, DC, USA, 2004.
[S05]G. Swamynathan et al. Decoupling Service and Feedback Trust in a Peer-to-Peer Reputation System. In Proceedings of International Workshop on Applications and Economics of Peer-to-Peer Systems, Nanjing, China, 2005.
[S06]G. Swamynathan et al. Exploring the Feasibility of Proactive Reputations. In Proceedings of International Workshop on Peer-to-Peer Systems, Santa Barbara, California, USA, 2006.
[T04]S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys, 36(4) pp.335–371, 2004.
[T66]R. M. Toms. Systems of Boolean Equations. The American Mathematical Monthly, Vol. 73, No. 1. pp. 29-35. 1966.
[V03]V. V. Vazirani. Approximation Algorithms. Springer, 2003. ISBN 3-540-65367-8.
[W01]A. N. Whitehead. Memoir on the Algebra of Symbolic Logic. American Journal of Mathematics, Vol. 23, No. 4. pp. 297-316.1901.
[W02]D. S. Wallach. A Survey of Peer-to-Peer Security Issues, 2002.
[Wa02]A. Wagner and B. Plattner. Peer-to-peer systems as attack platform for distributed denial-of-service. In ACM SACT Workshop, Washington, DC, USA, 2002.
[W04]K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In Proceedings of Recent Advances in Intrusion Detection , Sophia Antipolis, France 2004.
[W05]K. Wang and S. Stolfo. Anomalous payload-based worm detection and signature generation. In Proceedings of Recent Advances in Intrusion Detection, Seattle, Washington, USA, 2005.
[Y07]Y. Yan et al. Ranking-based Optimal Resource Allocation in Peer-to-Peer Networks. In Proceedings of INFOCOM'07, Anchorage, Alaska, USA, 2007.
[Z01]B.Y. Zhao et al. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report, University of California at Berkeley. 2001.